PHAR LAP

386|LIB UTILITY GUIDE

386|DOS-EXTENDER SDK

386 | LIB
Utility Guide

Phar Lap Software, Inc.
60 Aberdeen Avenue, Cambridge, MA. 02138
(617) 661-1510, FAX (617) 876-2972
dox@pharlap.com
tech-support@pharlap.com

First edition: February 1987.
Second edition: January 1991.

Table of Contents

Preface

This manual, *386 | LIB Utility Guide*, is for people who intend to create and include object library files with an application program. This manual presumes a familiarity with the development process and various Phar Lap programs, including 386 | LINK and LinkLoc. For books and manuals with that information, please see the Related Documentation and Books section at the end of this preface.

This manual has one chapter, Using 386 | LIB, which documents the command line operands and two appendices, 386 | LIB Command Line Switches and 386 | LIB Error Messages.

If, after you have read this book, you find that you have suggestions, improvements or comments that can make this a better manual, please call us, write us or send us mail at:

Phar Lap Software, Inc.
60 Aberdeen Avenue, Cambridge, MA. 02138
(617) 661-1510, FAX (617) 876-2972
dox@pharlap.com
tech-support@pharlap.com

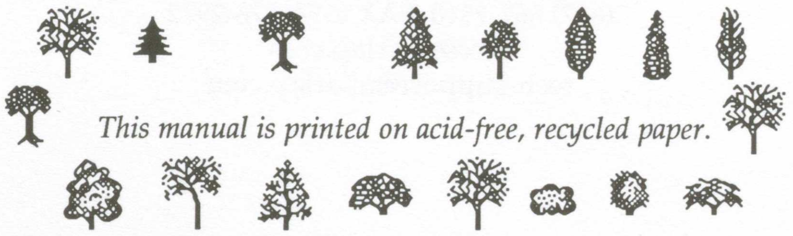
Manual Conventions

This manual relies on certain conventions to convey certain types of information. On the following pages, these are the conventions:

`courier`	indicates the items on the command line.
`italics`	indicate the items on the command line that must have a user-entered name or value in place of the items in italics.
`[]`	square brackets indicate that one or more of the enclosed items may be chosen. It is valid not to choose any of the items in square brackets.
`...`	ellipsis indicates an item or items that may be repeatedly entered.

Related Documentation and Books

Intel Corporation. *The Concrete Representation of 80286 Module Formats.*
Phar Lap Software, Inc., *386 | LINK Reference Manual.*



This manual is printed on acid-free, recycled paper.

This manual was produced on a Macintosh IIcx, using MS-Word. The examples illustrated in this manual were developed with 386 | LIB, version 3.0.

Chapter 1

Using 386 | LIB

1.1 Introduction

386 | LIB is a librarian for OMF–86 and Easy OMF–386 object files. It is a member of the 80386 Software Development Series from Phar Lap Software, Inc. 386 | LIB is used to create, modify, and inspect object library files which can be processed by the Phar Lap linkers 386 | LINK and LinkLoc. Versions of 386 | LIB which run on the IBM PC or PC/AT, VAX/VMS, and a number of different UNIX systems are available. This manual describes how to use 386 | LIB.

1.2 Command Line Syntax

The command line used to run the librarian is the name of the librarian task image (386LIB on IBM PC systems, XLIB386 on other systems), followed by a list of file names and switches. The switches are used to override the default operation of the librarian. When errors occur during processing of a library, an error message is displayed on the screen and written to the map file if it exists.

A file name can be specified on the command line as a complete file name (*filename.extension*) or can be given without an extension, in which case 386 | LIB supplies a default extension. The librarian assumes the following file name extensions when none are specified:

File Type	Default Extension on IBM PC MS-DOS	Default Extension on VAX/VMS and UNIX
object file	.OBJ	.O86
library file	.LIB	.L86
library backup file	.BAK	.BAK
library map file	.MAP	.MAP
librarian command file	.LBC	.LBC

In addition, a complete or partial path may be specified with the file name. If none is given, the current default device and directory are assumed.

Switches begin with the minus sign character (-) followed by the name of the switch. No spaces are permitted between the minus sign and the switch name. Any arguments to the switch must immediately follow the switch, with spaces as separators. Adjacent input file and module names may be separated by spaces and/or commas. Adjacent switches, or an adjacent switch and input file or module name, must be separated by spaces. Input file names and switches may be placed in any order on the command line. Input file and module names may not begin with a minus sign character, so that 386 | LIB can distinguish between file names and switches. Some example command lines intended to demonstrate the different 386 | LIB operations are included with the command line switch descriptions.

1.3 Indirect Command Files

Like all other Phar Lap products, it is possible to place commonly used 386 | LIB command line parameters in an indirect command file. 386 | LIB command line parameters are entered in an indirect file in the same manner as they would be entered on the command line. Command line parameters in indirect command files can be entered using as many lines as necessary. To use an indirect command file, preface the name of the file on the command line with an at sign (@) character. When 386 | LIB encounters a command line parameter which begins with an @ character, it will open the file and process the commands in it immediately. If the file name does not have an extension specified, a default of .LBC is used. For example, the command

```
    386LIB @comfile -add newmod
```

would cause the command line parameters in the file named COMFILE.LBC to be processed before the "-add newmod" parameters.

1.4 386 | LIB Environment Variable

The environment variable 386LIB contains a string of one or more command switches. This variable is always read by 386 | LIB: on MS-DOS and UNIX systems, 386 | LIB looks in the environment itself; on VMS systems, 386 | LIB looks in the logical name table and in the system name table.

Since the environment variable, 386LIB, is scanned before the command line is read, the switches specified in the variable can be overridden by specifying conflicting switches on the command line. 386 | LIB follows the rule that the rightmost (last processed) switch takes precedence.

The following MS-DOS SET command configures 386 | LIB to process user–defined symbols in a case-sensitive manner.

```
    set 386LIB=-twocase
```

1.5 Command Line Switches

Command line switches are used to change the default operation of 386 | LIB. By default, 386 | LIB will:

- Create an output library which has the same file name as the input library
- Process all symbol names with case insensitivity (upper and lower case versions of a symbol being considered identical)
- Use a library page size of 16 bytes
- Not create library map or backup files.

Command line switches begin with a minus sign character (-) followed by the name of the switch. There are two forms of each switch name: a long form and a short form. Any argument to the switch must immediately follow the switch name, with a space as a separator. If conflicting switches are given on the command line, the rightmost (last processed) switch takes precedence.

1.5.1 Object Module Switches

The object module switches are used to add, delete, and extract object modules from libraries. The switches which operate on object modules contained in files use the file name which contains the object module as an argument. The switches which operate on modules contained in the library use the module's name in the library as an argument. Section 1.7 describes the module–naming rules employed by 386 | LIB. Consult the description for a particular switch to see whether it uses file names or module names as switch arguments.

Note: Any of the switches which use object file names as arguments will also accept an object library as an argument. If an object library is specified as a switch argument, each module in the library will be processed as appropriate. This feature is useful for merging the contents of two library files.

The -ADD switch is used to add new object modules to an existing object library. The modules are specified by the arguments to the -ADD switch. If one of the modules being added is already in the library, 386 | LIB will post an error and the library will not be updated. If the specified library does not yet exist, 386 | LIB will post a warning and then create the library before adding the specified modules to it.

Syntax:

```
-ADD filename ...
```

Short Form:

```
-A filename ...
```

Example:

IBM PC/MS-DOS

```
386lib mlib -add newmod
386lib mlib -a \obj\nm.obj \newobj\mod2 \obj\mod3
```

VAX/VMS

```
xlib386 mlib -add mod1.o86
xlib386 mlib -a DRA1:[obj]mod1
```

UNIX

```
xlib386 mlib -add new_module
xlib386 mlib -a ../obj/mod1.o ../newobj/mod2
```

The -CREATE switch instructs 386 | LIB to create the library from scratch using the modules specified by the switch arguments. If the library file already exists, the current contents are discarded before creating the new library.

Syntax:

```
-CREATE filename ...
```

Short Form:

```
-C filename ...
```

Example:

IBM PC/MS-DOS

```
386lib newlib -create mod1.obj mod2.o mod3
386lib newlib -c \obj\mod1 \newobj\mod2 \obj\mod3
```

VAX/VMS

```
xlib386 newlib -create mod1.o86 mod2 mod3
xlib386 newlib -c DRA1:[obj]mod1 [-.newobj]mod2 mod3
```

UNIX

```
xlib386 newlib -create mod1.o86 mod2 mod3
xlib386 newlib -c ../obj/mod1 ../newobj/mod2 mod3
```

The -DELETE switch causes 386 | LIB to remove the specified modules from the library being processed. The -DELETE switch uses module names as arguments, not file names. This is because the arguments reference modules which have already been inserted into a library.

If the module or library does not exist, 386 | LIB will post an appropriate error.

Syntax:

```
-DELETE module ...
```

Short Form:

```
-D module ...
```

Example:

```
386lib mathlib -delete trigfunc
386lib mlib -d sin cos tan
```

The -EXTRACT switch causes 386 | LIB to extract a copy of one or more object modules from a library. It is followed by a list of module names to be extracted. Each module extracted is placed in a file whose name is the module name with the default object file extension appended to it. Extracting object modules from a library does not alter the original library in any way.

Syntax:

```
-EXTRACT module ...
```

Short Form:

```
-E module ...
```

Example:

```
386lib mathlib -extract trigfunc
386lib mlib -e sin cos tan
```

The -EXTRACTALL switch causes 386 | LIB to extract a copy of all the object modules contained in a library. Its operation is exactly the same as if the -EXTRACT switch were specified for each module in the library. Each module extracted is placed in a file whose name is the module name with the default object file extension appended to it. Like the -EXTRACT switch, the -EXTRACTALL switch does not alter the original library in any way.

Syntax:

```
-EXTRACTALL
```

Short Form:

```
-EALL
```

Example:

```
386lib mathlib -extractall
386lib mlib -eall
```

The -REPLACE switch is used to replace one or more object modules in an existing library. The switch is followed by the names of the files containing the object modules to be replaced in the library. If a module being replaced is already in the library, 386 | LIB will delete it and then add the new object module. If a module is not yet in the library, 386 | LIB will post a warning and then add the module to the library. If the specified library does not yet exist, 386 | LIB will post a warning and then create the library before adding the specified modules to it.

Syntax:

```
-REPLACE filename ...
```

Short Form:

```
-R filename ...
```

Example:

IBM PC/MS-DOS

```
386lib clib -replace newmod
386lib hce -r \newobj\mod2 \nobj3\newmod3.o
```

VAX/VMS

```
xlib386 mlib -replace [nassist.objrcs]newmod1.o86
xlib386 clibs -r DBA3:[newobj]mod1
```

UNIX

```
xlib386 mlib -replace new_module
xlib386 libm -r ../newobj/mod1 ../nobj/newmod2.o
```

1.5.2 Map File Switches

The -MAP switch causes 386 | LIB to create a library map file which contains a list of all the object modules in the library along with a list of all the public symbols defined in each module. The switch is followed by the map file name. If the specified map file name does not have an extension, 386 | LIB will assign a default of .MAP. Section 1.8 of this manual contains a description of the library map file format and includes some short examples.

Syntax:

```
-MAP filename
```

Short Form:

```
-M filename
```

Example:

```
386lib mathlib -map mathlib
386lib libc -m libc.lmp
```

The -NOMAP switch causes 386 | LIB to not create a library map for the library being processed. This is the default operation of 386 | LIB, and this switch is only included for completeness.

Syntax:

```
-NOMAP
```

Short Form:

```
-NOM
```

Example:

```
386lib mathlib -extract sin -nomap
386lib mlib -delete cos -nom
```

1.5.3 Library Backup File Switches

The -BACKUP switch causes 386 | LIB to create a backup copy of the library before performing an operation which modifies it. The name of the backup file is the base name of the library file with the .BAK extension appended to it.

Syntax:

```
-BACKUP
```

Short Form:

```
-B
```

Example:

```
386lib libc -replace printf -backup
386lib mlib -delete cos -b
```

The -NOBACKUP switch causes 386 | LIB to not create a backup copy of the library before performing an operation which modifies it. This is the default operation of 386 | LIB; the switch is only included for completeness.

Syntax:

```
-NOBACKUP
```

Short Form:

```
-NOB
```

Example:

```
386lib libc -replace printf -nobackup
386lib mlib -delete cos -nob
```

1.5.4 Library Page Size Switch

The -PAGESIZE switch is used to specify the library page size of a library file. If the -PAGESIZE switch is not specified, 386 | LIB uses the current page size of the library being processed. If the library is being created and the -PAGESIZE switch is not specified, 386 | LIB assumes a default library page size of 16 bytes. This page size is used when aligning object modules in the library, and also limits the maximum size of the library. A bigger page size permits a larger maximum library size, but wastes more space between the object modules in the library. The -PAGESIZE switch parameter must be a power of 2 between 16 and 512 inclusive. The maximum library sizes for each of the permissible switch values are:

Page Size	Maximum Library Size
16	1 Megabyte
32	2 Megabytes
64	4 Megabytes
128	8 Megabytes
256	16 Megabytes
512	32 Megabytes

The library page size parameter is used whenever an operation is performed which modifies the library. Thus, it is possible to change the page size for an existing library file by using the -PAGESIZE switch while performing an operation which modifies the file.

Syntax:

```
-PAGESIZE size
```

Short Form:

```
-PAGESIZE size
```

Example:

To create a library with a page size of 64 bytes:

```
386lib newlib -create mod1.obj mod2.o mod3 -pagesize 64
```

To change the page size of a library named "small.lib" to 128 bytes, calling the new library "large.lib":

```
386lib large -create small -pagesize 128
```

1.5.5 Case Sensitivity Switches

By default, 386 | LIB is insensitive to the case of user-defined symbols. For example, the symbols "my_sym", "my_SYM", and "MY_SYM" are all considered identical by 386 | LIB. A command line switch can be used to make 386 | LIB process symbols in a case-sensitive manner. If this option is enabled, the three symbols in the above example would all be considered different.

The -ONECASE switch disables case-sensitive processing of user-defined symbols. This is the default mode of 386 | LIB; this switch is, therefore, redundant but is provided for consistency.

Syntax:

```
-ONECASE
```

Short Form:

```
-OC
```

Example:

```
386lib slibc -replace scanf -onecase
386lib mlib -add cos -tc
```

The -TWOCASE switch enables case sensitive processing of user-defined symbols. When this switch is used, upper and lower case versions of the same symbol are considered to be different.

Syntax:

```
-TWOCASE
```

Short Form:

```
-TC
```

Example:

```
386lib libc -replace printf -twocase
386lib mlib -add cos -tc
```

1.6 Order of Librarian Operations

When 386 | LIB is performing multiple operations on a library, it performs them in a predetermined order. A user does not usually need to be concerned with this order. The order of librarian operations is:

1. If the object library does not yet exist and it must be created, 386|LIB will create it

2. Any modules which must be extracted from an existing library are extracted before the library is modified

3. Any modules which are being deleted or replaced are removed from the library

4. The modifications to the library are completed by adding any new modules to the library. These modules include any arguments to the -ADD, -CREATE, and -REPLACE switches

5. Lastly, the library map file is created if the -MAP switch was specified.

1.7 Module Naming Conventions

Whenever 386 | LIB adds an object module to a library, it sets the module name record in the object module to the base name of the file containing the object module. This feature is implemented for compatibility with the Microsoft librarian and is useful when a language translator program does not generate correct module name information.

For example, a module contained in the DOS file

```
c:\386obj\startup.obj
```

is added to a library with its module name record set to "startup", independent of the original name the object module had.

386 | LIB can process object libraries created by other librarians which use the same library file format but different module naming conventions. When 386 | LIB encounters an object module which does not follow its naming conventions, it will extract the base file name from the module name record and allow that name to be used with any of the switches

requiring a module name. Thus an object module with a module name record of

```
d:\obj\new\recalc.asm
```

would be referenced by using the name "recalc" with any of the switches requiring a module name.

1.8 Library Map Description

The map file created by the librarian contains a list of the object modules in the library, along with a list of the public symbols defined in each module. A header at the top of the map identifies the 386 | LIB version number and the name of the library being processed. The public symbol list for each module is ordered in the same way as the symbols are defined in the module. There is no information about the location of each routine in its module, as this information is calculated at link time. The following example shows the library map file for a small library called "test.lib", which is made up of two object modules called "init.obj" and "testmain.obj":

```
386|LIB: 3.0 -- Copyright (C) 1986-90 Phar Lap Software, Inc.
Library File - test.lib

Module:  init
    _start
    _exit

Module:  testmain
    _main
    _sub1
    _recalc
    _redisp
    _makecell
    _delcell
```

1.9 Converting Intel OMF–86 Libraries

386 | LIB can convert Intel style OMF-86 libraries into Phar Lap style OMF-86 libraries for input to 386 | LINK or LinkLoc. An example command line which performs this conversion follows:

```
386LIB pharlap.lib -create intel.lib
```

Appendix A

386 | LIB Command Line Switches

386LIB	*libname filename* ...	Library name [.LIB]
	@*file*	Open indirect command file [.LBC]
	-ADD *filename* ...	Add new object modules to a library
	-BACKUP	Make a backup library file
	-CREATE *filename* ...	Create a library from object files
	-DELETE *module* ...	Delete object modules from a library
	-EXTRACT *module* ...	Extract object modules from a library
	-EXTRACTALL	Extract all object modules from a library
	-MAP *filename*	Create a library map file [.MAP]
	-NOBACKUP	Do not make a library backup file
	-NOMAP	Do not create a library map file
	-ONECASE	Case-insensitive symbol processing
	-PAGESIZE *number*	Use a library page size of *number*
	-REPLACE *filename* ...	Replace object modules in a library
	-TWOCASE	Case-sensitive symbol processing

This page intentionally left blank.

386 | LIB Error Messages

If an error occurs during library processing, 386 | LIB displays an error message on the screen, which identifies the error which occurred along with its severity. If the error is associated with a module and/or user-defined symbol, the name of the module and/or symbol is displayed as well. The same message is also copied to the library map file if a map file is created.

Errors are divided into three categories: warning errors, severe errors, and fatal errors. Warning errors are errors which do not compromise the integrity of the library and can be ignored if desired; all other specified operations will be performed on the library. Severe and fatal errors are errors which would result in an invalid library being created. If a severe or fatal error occurs, the library file being processed is left unchanged.

The error message descriptions which follow are organized into sections which correspond to the three different categories of errors. The error messages are listed in alphabetical order within each of these sections.

B.1 Warning Errors

`Warning:`	`Library file "XXX" does not exist so it will be created`
Cause:	The librarian was instructed to add object modules to a library file that does not exist. When this occurs, 386 | LIB automatically creates the library and then adds the specified modules to it.
Solution:	Since this is a warning error, it can be ignored. To avoid the warning, use the -CREATE option to create the library from the object modules instead.
`Warning:`	`Module "XXX" not found -- delete ignored`
Cause:	An attempt was made to delete a module which is not present in the library.
Solution:	Check the spelling of the module name against the names of the modules in the library.
`Warning:`	`Module "XXX" not found -- extract ignored`
Cause:	An attempt was made to extract a module which is not present in the library.
Solution:	Check the spelling of the module name against the names of the modules in the library.

`Warning:`	`Module "`*`XXX`*`" not found -- module was added, not replaced`
Cause:	An attempt was made to replace a module which is not present in the library.
Solution:	Since this is a warning error, it can be ignored. Otherwise, check the spelling of the module name against the names of the modules in the library.

B.2 Severe Errors

`Error:`	`Because of errors, library file "`*`XXX`*`" has not been updated`
Cause:	This message is displayed along with other severe errors to indicate that the library file being processed was left unchanged because the errors would have caused an invalid library file to be created.
Solution:	Correct all of the other severe errors signaled by the librarian.

Error: `Cannot create object file for module "`*`XXX`*`" -- invalid file name.`

Cause: The file name, which 386 | LIB would use for a module being extracted from a library, is not valid. 386 | LIB creates this file name by using the name of the module in the library and appending the default object file extension. It is possible to cause this error by transferring a library between hosts which allow different special characters in file names.

Solution: The only solution to this problem is to extract the desired module(s) from the library on a host which allows the same file names as the host on which the library was created.

Error: `Duplicate definition of symbol "`*`XXX`*`" in file "`*`YYY`*`"`

Cause: A public symbol is defined in more than one object module.

Solution: 1) Rename the public symbol so that it does not conflict with the other public symbol

2) Delete the public symbol from the module

3) Do not attempt to put one of the object modules defining the symbol in the library.

`Error:` `Module "XXX" is already in the library`

Cause: An attempt has been made to add an object module to a library and another module with the same name is already present in the library.

Solution: 1) Rename the object file to a different name

2) Use the -REPLACE switch instead of the -ADD switch to delete the old object module from the library before adding in the new object module.

B.3 Fatal Errors

`Fatal Error:` `Bad library file "XXX"`

Cause: The specified library file was not valid.

Solution: Make sure that the specified file conforms to either Phar Lap or Microsoft library file format.

`Fatal Error:` `Bad number specified on the command line -- "XXX"`

Cause: An invalid number was specified as a command switch argument. This error will occur if the number is badly formed (illegal numeric characters), or if it is out of range for the switch.

Solution: Check the number to make sure it is valid and within the permissible range for the switch.

```
Fatal Error: Bad object module in file "XXX"
```

Cause: An 386 | LIB operation was attempted with an invalid object module.

Solution: 1) Make sure that the object module is a valid OMF–86 or Easy OMF–386 file

2) If any errors occurred when compiling or assembling the file, correct the errors before attempting to add the object file to a library

3) It is possible to use the Phar Lap linker (386 | LINK or LinkLoc) to get a more detailed message about what is wrong with the object file by linking the file by itself. The linker should display the same "invalid object file" message, followed by the reason the file is invalid.

```
Fatal Error: Cannot create "XXX" file -- reason
```

Cause: 386 | LIB was unable to create the file named "*XXX*" due to a host operating system error. The reason for the error is displayed at the end of the message and is host dependent.

Solution: Consult the host operating system documentation to clarify the meaning of the error and take appropriate action to clear it.

`Fatal Error: Cannot open "`*`XXX`*`" file -- `*`reason`*

Cause: 386 | LIB was unable to open the file named *"XXX"* due to a host operating system error. The reason for the error is displayed at the end of the message and is host dependent.

Solution: Consult the host operating system documentation to clarify the meaning of the error and take appropriate action to clear it.

`Fatal Error: Command too long -- "`*`XXX`*`"`

Cause: A command line parameter specified in an indirect command file is longer than 255 characters.

Solution: Shorten the parameter which is too long.

`Fatal Error: Error reading from "`*`XXX`*`" file -- `*`reason`*

Cause: 386 | LIB was unable to read data from the file named *"XXX"* due to a host operating system error. The reason for the error is displayed at the end of the message and is host dependent.

Solution: Consult the host operating system documentation to clarify the meaning of the error and take appropriate action to clear it.

`Fatal Error: Error writing to "XXX" file -- reason`

Cause: 386 | LIB was unable to write data to the file named *"XXX"* due to a host operating system error. The reason for the error is displayed at the end of the message and is host dependent.

Solution: Consult the host operating system documentation to clarify the meaning of the error and take appropriate action to clear it.

`Fatal Error: Extra command line argument -- "XXX"`

Cause: More than one library file was specified on the command line, or a switch argument was specified for a switch which does not use one.

Solution: Check the command line parameters to eliminate the extraneous argument.

`Fatal Error: Invalid file name -- "XXX"`

Cause: A file name was specified on the command line which does not follow the correct syntax for file names on the host computer.

Solution: Use a valid file name.

```
Fatal Error: Library file has overflowed -- increase page size
```

Cause: The size of the library file has exceeded 65,536 times the library page size.

Solution: Use the -PAGESIZE switch to specify a larger page size for the desired operation. Once the library has been successfully updated using the new page size, it is no longer necessary to specify the -PAGESIZE switch until the file overflows again.

```
Fatal Error: Missing value to command switch -- "XXX"
```

Cause: A 386 | LIB command switch which requires an argument was specified without an argument.

Solution: Specify a value for the switch.

```
Fatal Error: One command file cannot call another command
             file -- "XXX"
```

Cause: An attempt has been made to open an indirect command file inside another indirect command file. 386 | LIB does not support nesting of indirect command files.

Solution: 1) Combine the two command files into a single command file.

2) Open the second indirect command file on the 386 | LIB command line, rather than inside the first indirect command file.

```
Fatal Error:  Unknown command switch -- "XXX"
```

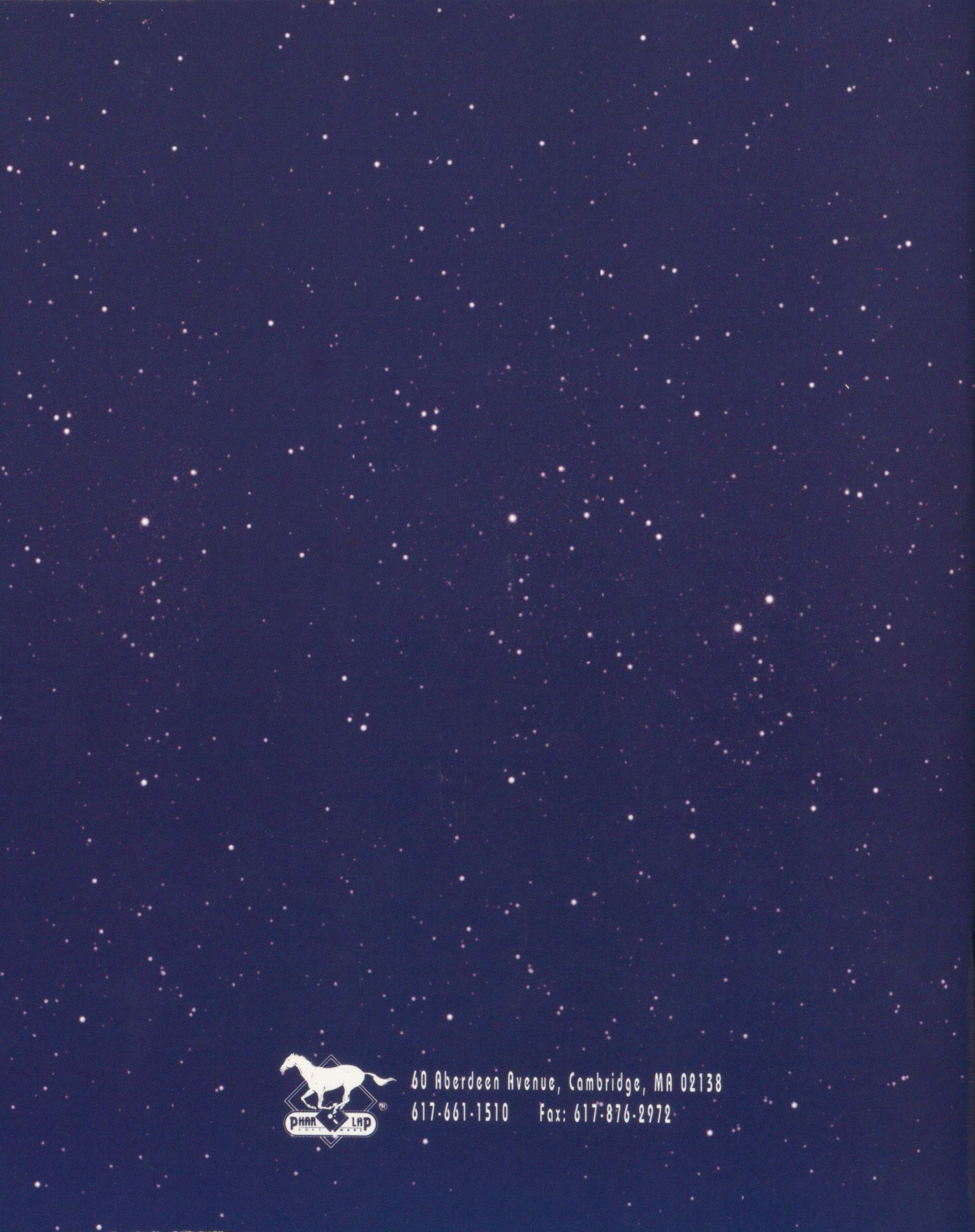
Cause: A command switch which was not recognized by 386 | LIB was specified on the command line.

Solution: Check the spelling of the switch name against the list of valid 386 | LIB switches.

Index

The people who wrote, edited, revised, reviewed, indexed, formatted, polished, and printed this manual were:

John Benfatto, Alan Convis, Noel Doherty, Lorraine Doyle, Bryant Durrell, Nan Fritz, Elliot Linzer, John Mann, Kim Norgren, Cate O'Hara, Richard Smith, Amy Weiss, and Rick Wesson.



PHAR LAP SOFTWARE®

60 Aberdeen Avenue, Cambridge, MA 02138

617-661-1510 Fax: 617-876-2972